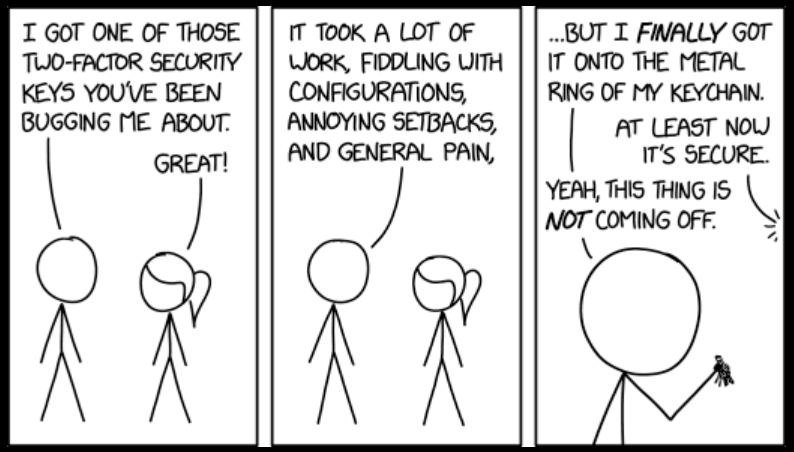
Hardware Security

ECEn 522R

# Modern CPU Extensions for Security (Part 2): CHERI and CHERIoT




Stacey D. Son 20-Nov-2025

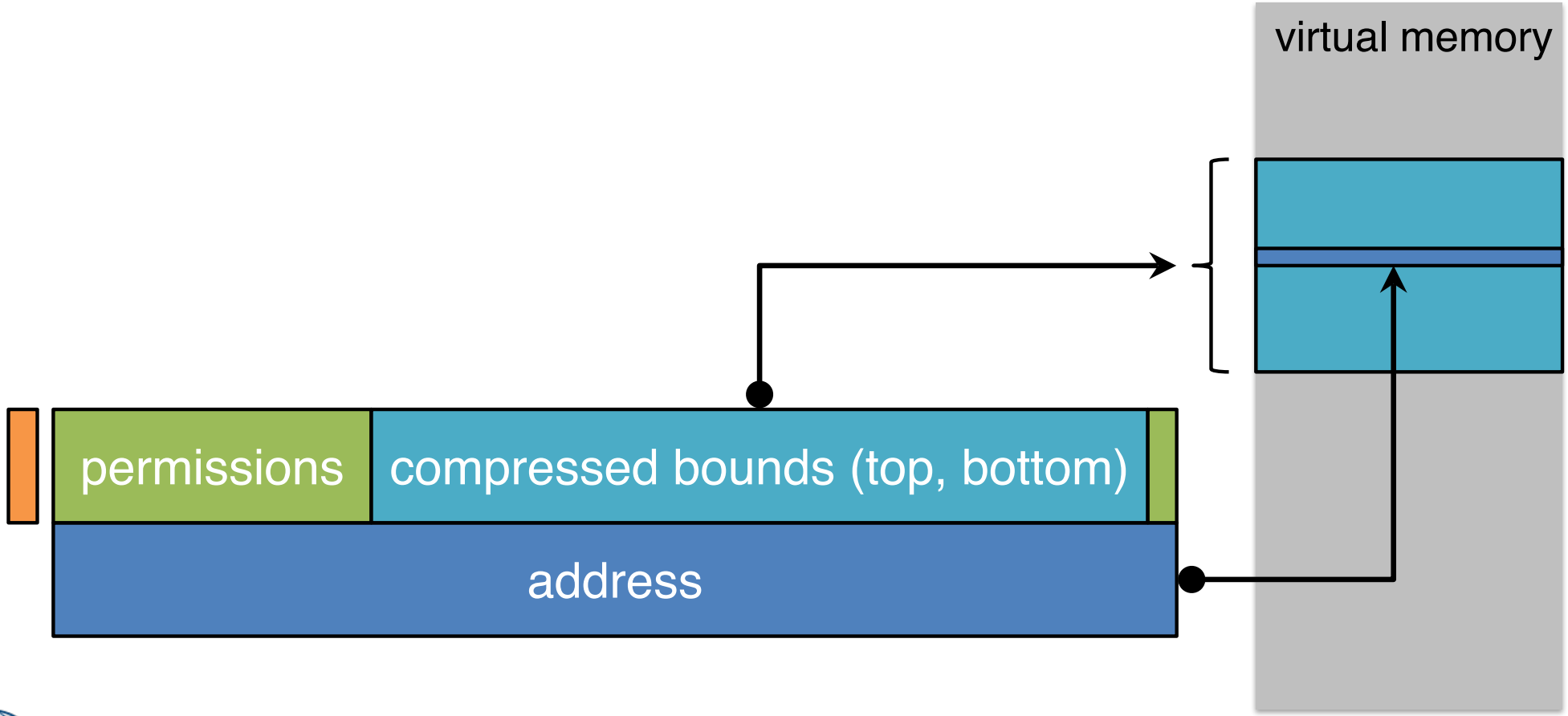
# Capability-based Security

- A Capability is communicable, unforgeable token of authority.
- A simple example of a capability in software:
  - A privileged process opens a file and passes the file descriptor (via IPC) to another process running with less privilege. The file descriptor is a token of authority and gives the less privileged process access to the secure file.
  - Capsicum extends this idea of using file descriptors as capability tokens for FreeBSD and Linux limiting name spaces in the kernel (i.e., sandboxing).
- Capabilities enforce the *Principle of Least Privilege* by allowing a program to access only the things it needs and nothing more.
- Capabilities also may enforce the *Principle of Intentional Use*. For example, while an agent may have access to more that it needs a capability limits to its intentional use. (Prevents “confused deputy” problems.)

# Memory Capabilities

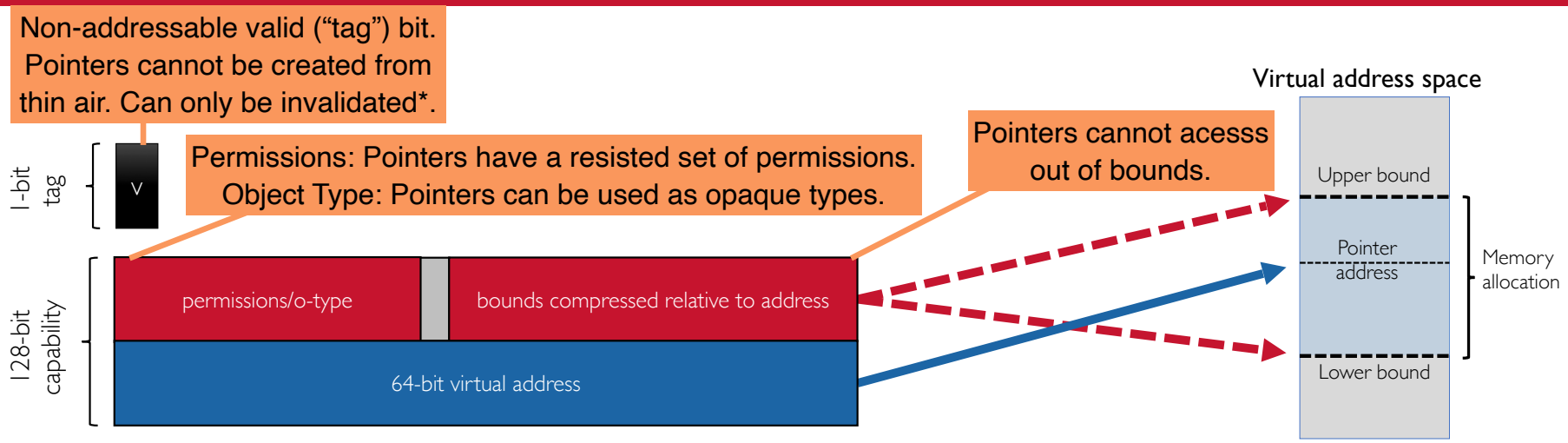
- A token for accessing memory
  - Gives access to a region of memory
  - Gives permissions to memory: read, write, execute, if can read and write capabilities, etc.
  - They can not be forged (e.g., by writing an integer to memory)
  - Easily invalidated by overwriting (like with another type)
- Example: Memory Allocation
  - Allocator will returns a capability grants access to memory requested

# A atomic new type – the **Capability**



# Capability Hardware Enhanced RISC Instructions (CHERI)

# CHERI capability type




- **Capabilities** extend **integer memory addresses**
- **Metadata** (bounds, permissions, …) control how they may be used
- **Guarded manipulation** controls how capabilities may be manipulated; e.g., **provenance validity** and **monotonicity**
- **Tags** protect capability integrity/derivation in registers + memory

The "capability" of a capability can only be decreased.

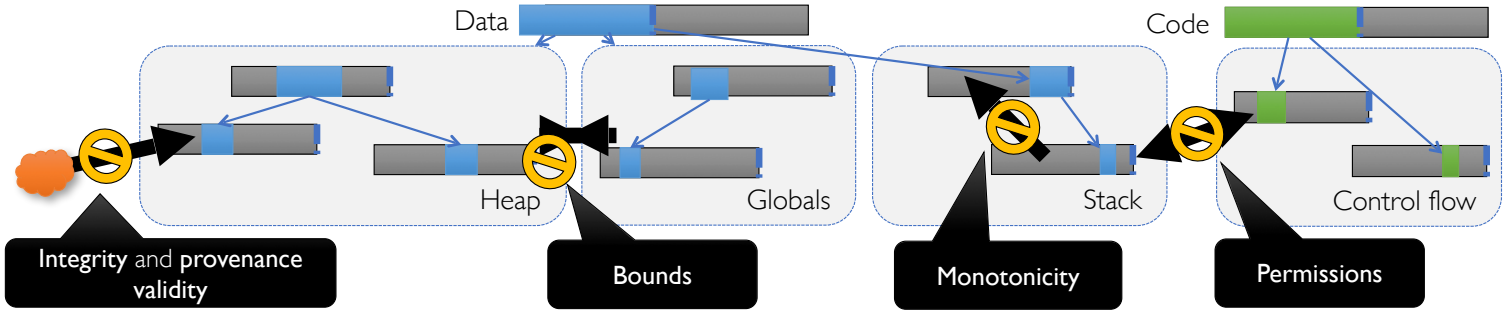
*Overwriting a capability in memory with a non-capability store invalidates tag bit.

CHERI

SRI

UNIVERSITY OF CAMBRIDGE

# CHERI enforces protection semantics for pointers



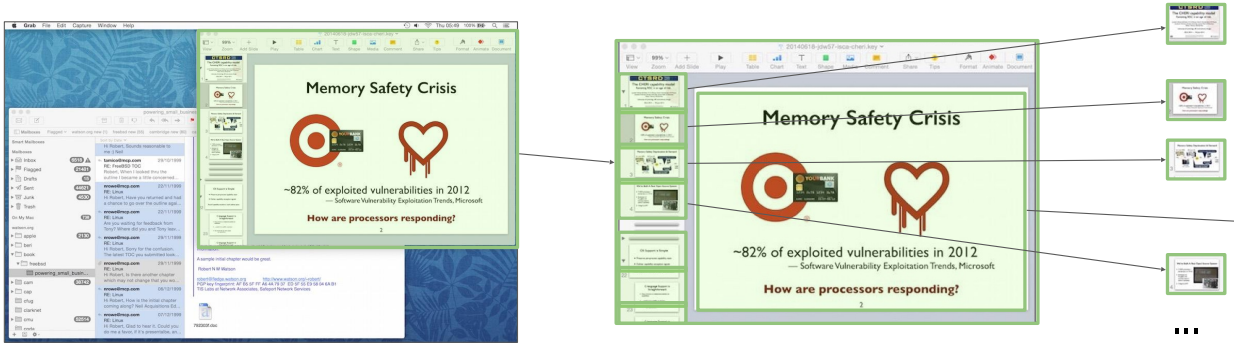

- **Integrity** and **provenance validity** ensure that valid pointers are derived from other valid pointers via valid transformations; **invalid pointers cannot be used**
- **Bounds** prevent pointers from being manipulated to access the wrong object
- **Monotonicity** prevents pointer privilege escalation – e.g., broadening bounds
- **Permissions** limit unintended use of pointers; e.g., W^X for pointers
- These primitives not only allow us to implement **strong spatial and temporal memory protection**, but also higher-level policies such as **scalable software compartmentalization**

CHERI



SRI UNIVERSITY OF CAMBRIDGE

# Software compartmentalization at scale




CHERI contains attack within compartment, preventing access to other data


- Current CPUs limit:
  - The number of compartments and rate of their creation/destruction
  - The frequency of switching between them, especially as compartment count grows
  - The nature and performance of memory sharing between compartments
- CHERI is intended to improve each of these – **by at least an order of magnitude**

CHERI compartments reduce the "blast radius" and can add fault recovery.

CHERI

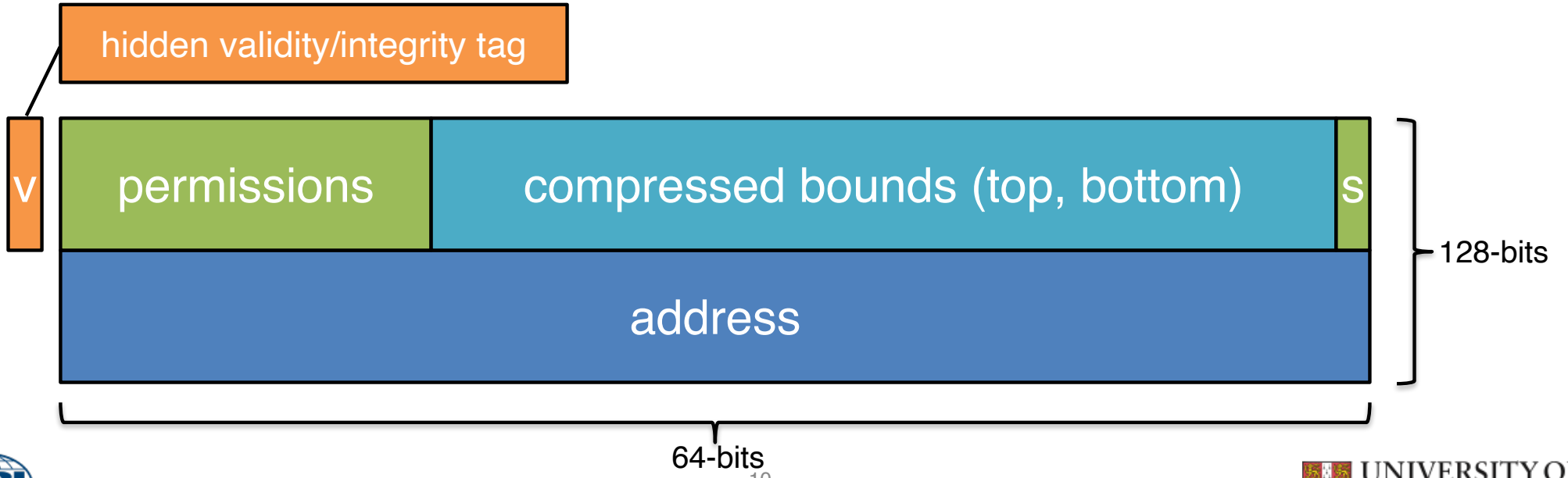


SRI UNIVERSITY OF CAMBRIDGE

So how does CHERI work?
(Let’s drill down)

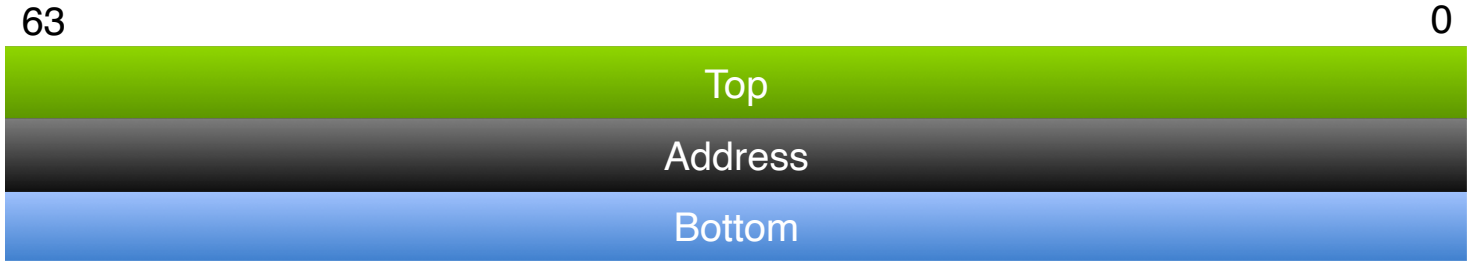
# The **Capability** type

- CHERI Capability = bounds checked pointer with integrity
- Held in memory and in (new or extended) registers

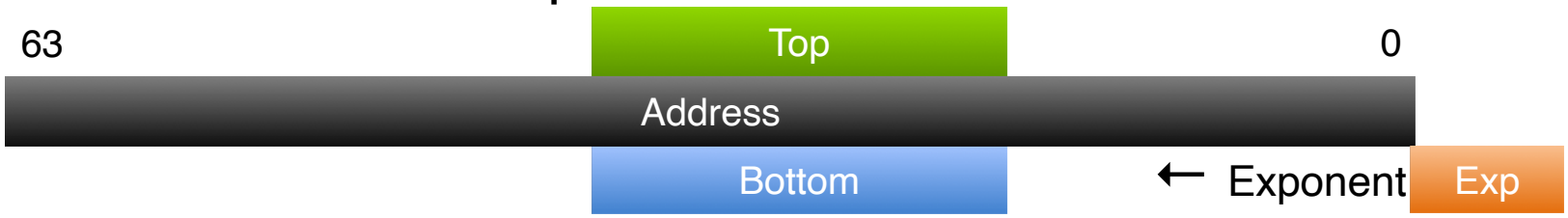


# Capability Compression

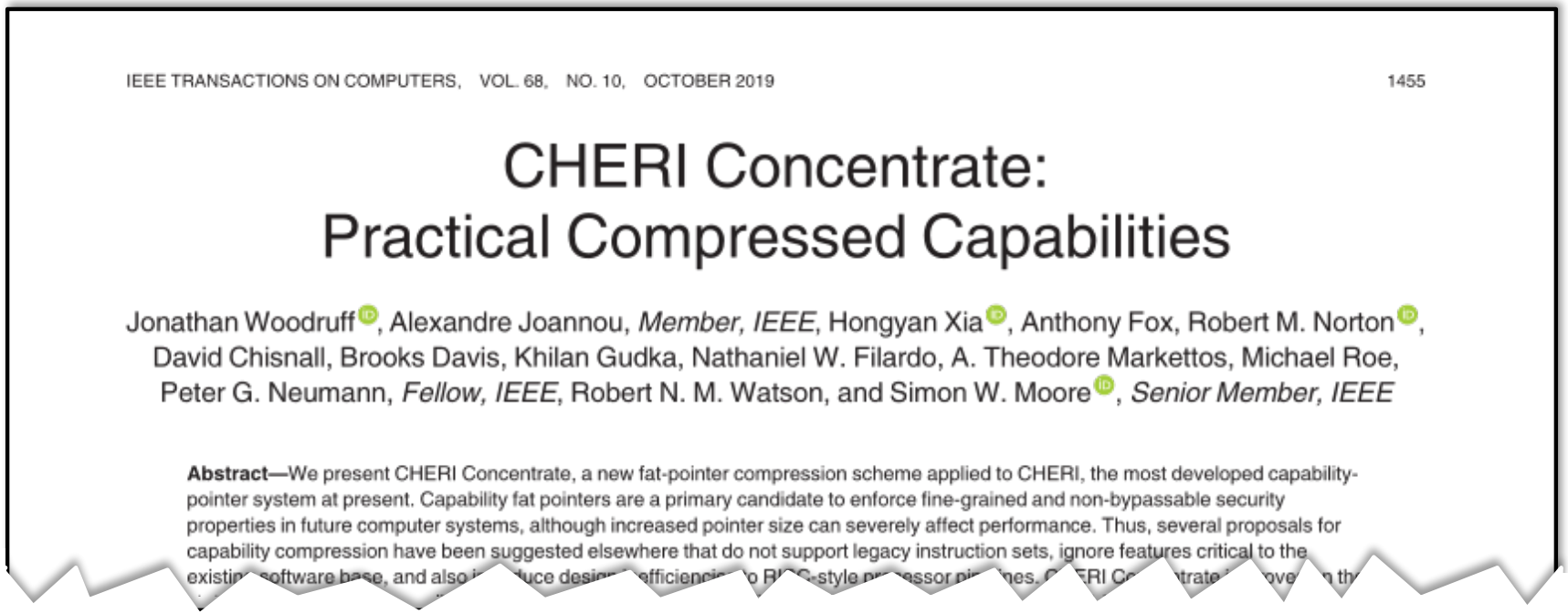
Capabilities encode three 64-bit fields (plus permissions, etc.):




But we can encode the Top and Bottom relative to the Address:




- Larger objects require greater alignment
- Address must be “near” the Top and Bottom



IEEE TRANSACTIONS ON COMPUTERS, VOL. 68, NO. 10, OCTOBER 2019 1455

# CHERI Concentrate: Practical Compressed Capabilities

Jonathan Woodruff, Alexandre Joannou, *Member, IEEE*, Hongyan Xia, Anthony Fox, Robert M. Norton, David Chisnall, Brooks Davis, Khilan Gudka, Nathaniel W. Filardo, A. Theodore Markettos, Michael Roe, Peter G. Neumann, *Fellow, IEEE*, Robert N. M. Watson, and Simon W. Moore, *Senior Member, IEEE*

**Abstract**—We present CHERI Concentrate, a new fat-pointer compression scheme applied to CHERI, the most developed capability-pointer system at present. Capability fat pointers are a primary candidate to enforce fine-grained and non-bypassable security properties in future computer systems, although increased pointer size can severely affect performance. Thus, several proposals for capability compression have been suggested elsewhere that do not support legacy instruction sets, ignore features critical to the existing software base, and also introduce design inefficiencies to RISC-style processor pipelines. CHERI Concentrate improves on th

- Published in IEEE Transactions on Computers, October 2019
- Has all the maths and formal proof in it…

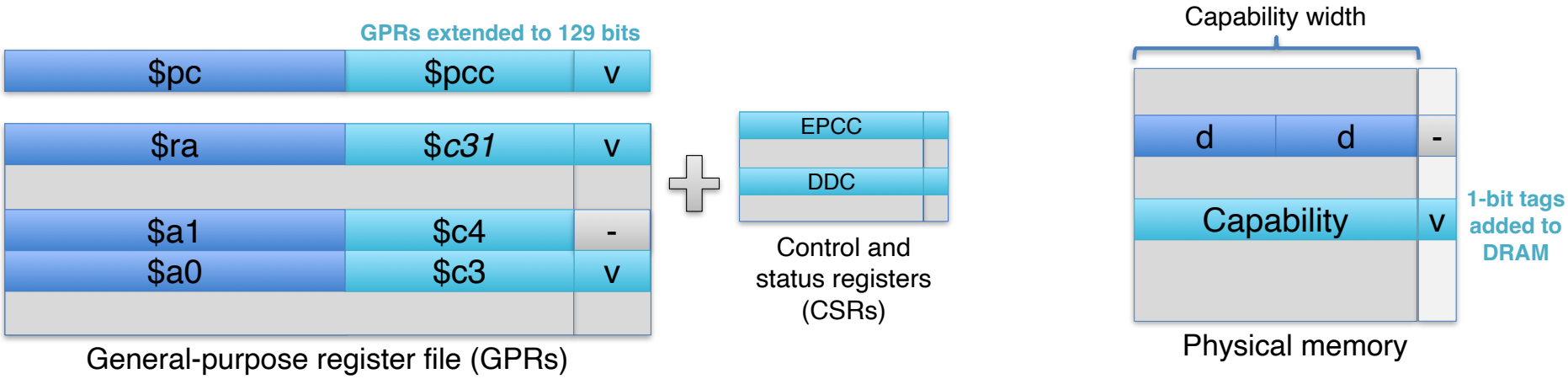
# Permissions

- **Permit_Execute (EX)** Allow this capability to be used in the PCC register as a capability for the program counter.
- **Permit_Store** (**SD)** Allow this capability to be used as a pointer for storing data from general-purpose registers.
- **Permit_Load (LD)** Allow this capability to be used as a pointer for loading data into general-purpose registers.
- **Permit_Store_Capability (SC)** Allow this capability to be used as a pointer for storing other capabilities.
- **Permit_Load_Capability (LC)** Allow this capability to be used as a pointer for loading other capabilities.
  **Permit_Load_Store_Capability (MC)** - Used with SD and LD
- **Permit_Store_Local_Capability (SL)** Allow this capability to be used as a pointer for storing local capabilities.
- **Global (GL)** Allow this capability to be stored via capabilities that do not themselves have *Permit_Store_Local_Capability* set.
- **Permit_Seal (SE)** Allow this capability to be used to seal or unseal capabilities that have the same **otype**. **Permit_Unseal (US)**
- **Access_System_Registers (SL)** Allow this capability to change system registers.
- **User_Perm0 (U0**) Software defined permission.

| Value | Name |
|---|---|
| 0 | Global |
| 1 | Permit_Execute |
| 2 | Permit_Load |
| 3 | Permit_Store |
| 4 | Permit_Load_Capability |
| 5 | Permit_Store_Capability |
| 6 | Permit_Store_Local_Capability |
| 7 | Permit_Seal |
| 8 | *reserved* User_Perm0 |
| 9 | *reserved* |
| 10 | Access_System_Registers |

# Merged capability register file + tagged memory

(as found in CHERI-RISC-V and ARM Morello)




- **64-bit general-purpose registers (GPRs)** are extended with **64 bits of metadata** and a **1-bit validity tag**
- **Program counter (PC)** is extended to be the **program-counter capability ($PCC)**
- **Default data capability ($DDC)** constrains legacy integer-relative ISA load and store instructions

Legacy Integer Support

- **Tagged memory** protects capability-sized and -aligned words in DRAM by adding a **1-bit validity tag**
- **Various system mechanisms** are extended (e.g., capability-instruction enable control register, new TLB/PTE permission bits, exception code extensions, saved exception stack pointers and vectors become capabilities, etc.)

# Low-level CHERI software models


**Unmodified**
All pointers are integers

**Hybrid**
Annotated and automatically selected pointers are capabilities

**Pure-capability** *
All pointers are capabilities

- **Source and binary compatibility**: **C-language idioms,** multiple **ABIs**
  - **Unmodified code**: Existing code runs without modification
  - **Hybrid code**: E.g., used in return addresses, for annotated data/code pointers, for specific types, stack pointers, etc.

    … But "hybrid" is a spectrum: many different choices for manual and automatic selection of integers vs. capabilities, API and ABI impacts
  - **Pure-capability code**: Ubiquitous data- and data-pointer protection. Not interoperable with legacy code due to changed pointer size.
- **CHERI Clang/LLVM compiler prototype** generates code for all three
- **CheriBSD** supports all three. Legacy binaries run fine without modification.



* CHERIoT Supports only Pure-capability

# New Instructions

- Memory access
  - Loads and stores via a bounds checked capability
  - Exception if address is out of range
- Guarded manipulation of capabilities
  - Decrease bounds
  - Decrease permissions
  - Adjust the address
  - Extract/test fields

monotonic decrease in rights guaranteed by formally verified hardware

critical property for security

# RISC-V vs. CHERI-RISC-V generated code

```
struct timezone tz;


time_t get_unix_time(void)
{
     struct timeval tv;

     gettimeofday(&tv,
&tz);
     return tv.tv_sec;
}
```

```
get_unix_time_riscv:
 addi    sp, sp, -32
 sd  ra, 24(sp)
 addi    a0, sp, 8
.LBB0_1:
 auipc   a1, %pcrel_hi(tz)
 addi    a1, a1, %pcrel_lo(.LBB0_1)
 call    gettimeofday
  (expands to auipc, possibly cld, cjalr)
 ld  a0, 8(sp)
 ld  ra, 24(sp)
 addi    sp, sp, 32
 ret
```

```
get_unix_time_cheririscv:
 cincoffset  csp, csp, -32
 csc     cra, 16(csp)
 cincoffset  ca0, csp, 0
 csetbounds  ca0, ca0, 16
.LBB0_1:
 auipcc  ca1, %captab_pcrel_hi(tz)
 clc ca1, %pcrel_lo(.LBB0_1)(ca1)
.LBB0_2:
 auipcc  ca2, %captab_pcrel_hi(gettimeofday)
 clc ca2, %pcrel_lo(.LBB0_2)(ca2)
 cjalr   cra, ca2
 cld     a0, 0(csp)
 clc     cra, 16(csp)
 cincoffset  csp, csp, 32
 cret
```

- The general code structure is unchanged except that:
  - The integer stack pointer becomes a capability stack pointer
  - The pointer to a local stack allocation becomes capability
  - Compiler-specified bounds are set on the local variable pointer before use
  - The loaded jump target is a capability rather than an integer address

1. Adjust stack address/capability
2. Save return address/capability
3. Create address/capability to local ‘tv’

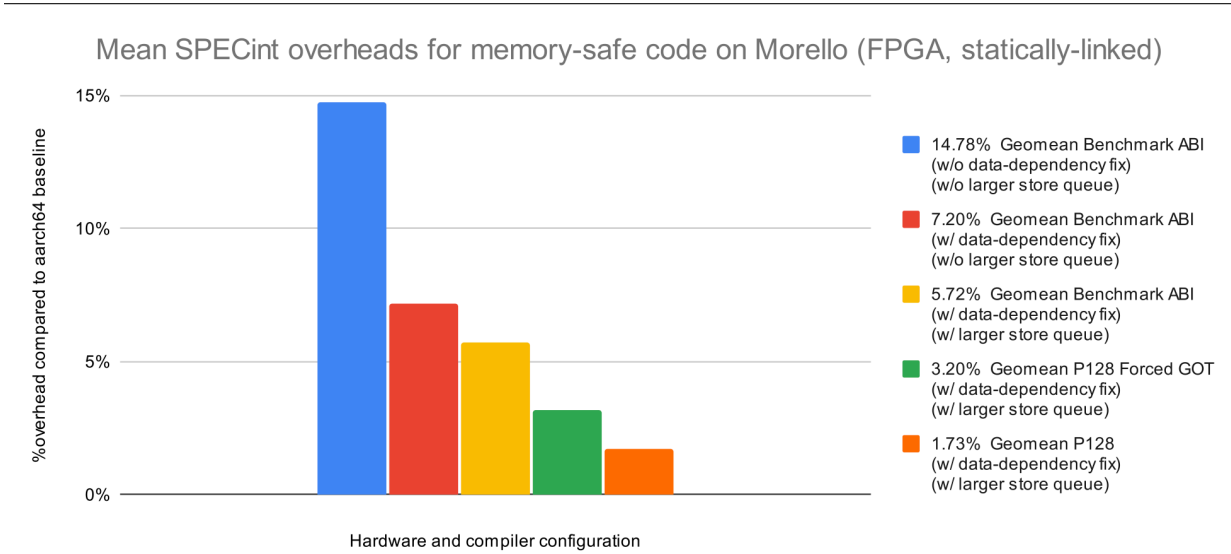
4. Generate address/capability to global ‘tz’

5. Call gettimeofday()

6. Load return value from ‘tv’
7. Load return address/capability
8. Restore stack address/capability
9. Return

# CHERI Overhead

- "Using an FPGA implementation with post-tapeout re-tuning, and compensating through ABI change for a microarchitectural issue affecting capability bounds prediction, the best measured over for a CHERI-adapted, spatially protected subset of SPECint 2006 running on Morello was **5.7%**."
- "Measured using a 128-bit pointer compilation to the 64-bit ISA on the Neoverse N1 microarchecture, the estimated overhead range for a more mature implementation was between **1.82% and 2.98%**."




Early results from the ARM Morello prototype and predicted results with further microarchecture optimizations.

https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/20240419-ieeesp-cheri-memory-safety.pdf

CHERI

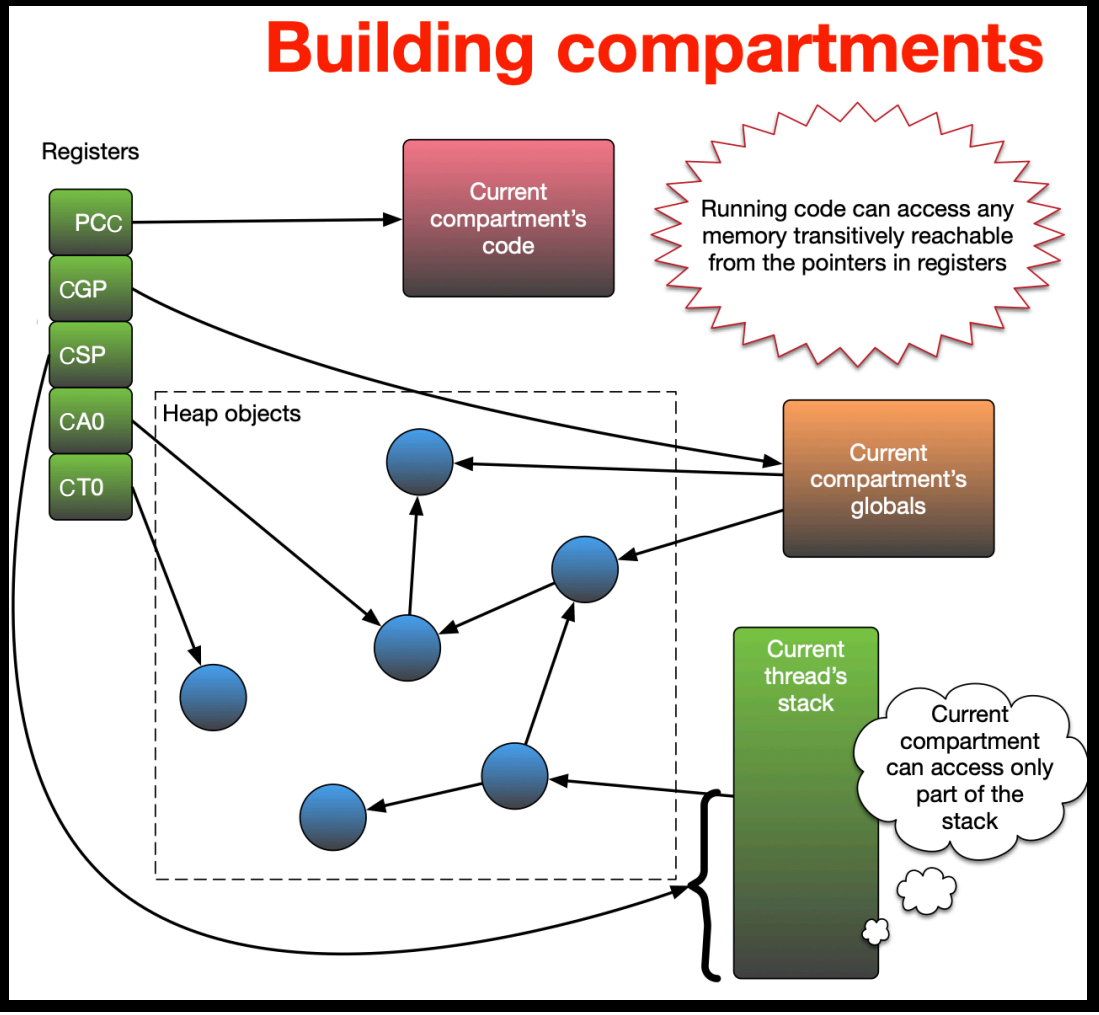
SRI

UNIVERSITY OF CAMBRIDGE

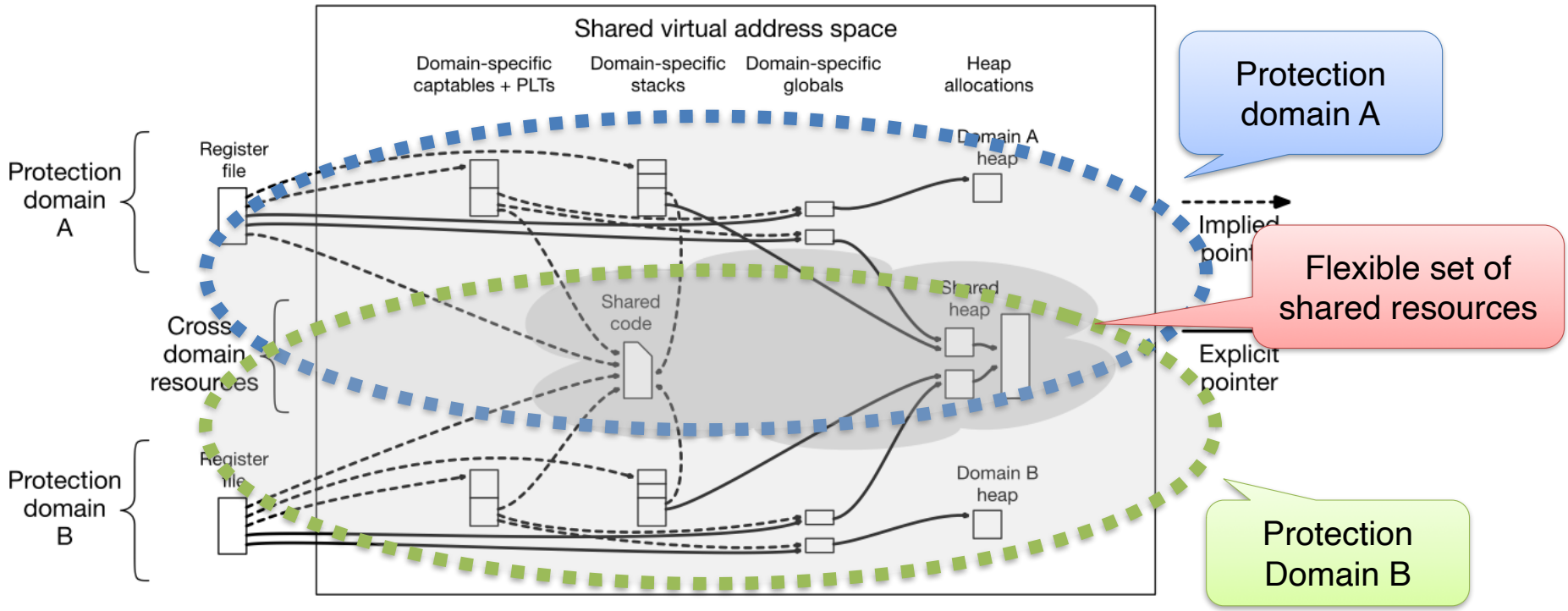
# CHERI Compartments

Compartments are invoked with a compartment switcher (about 200 instructions). At the end of a compartment transition, the new compartment has access to:

- Its own code (PCC).
- Its own globals (CGP).
- A portion of the thread's stack, excluding any frames owned by the caller, and full of zeros.
- Any memory pointed to by argument capability registers, passed explicitly from the caller.
- On return back from the compartment the caller only has access to the explicit return capabilities.



# CHERI-based compartmentalization sharing




- Isolated compartments can be created using closed graphs of capabilities, combined with a constrained non-monotonic domain-transition mechanism

# Compartmentalization scalability

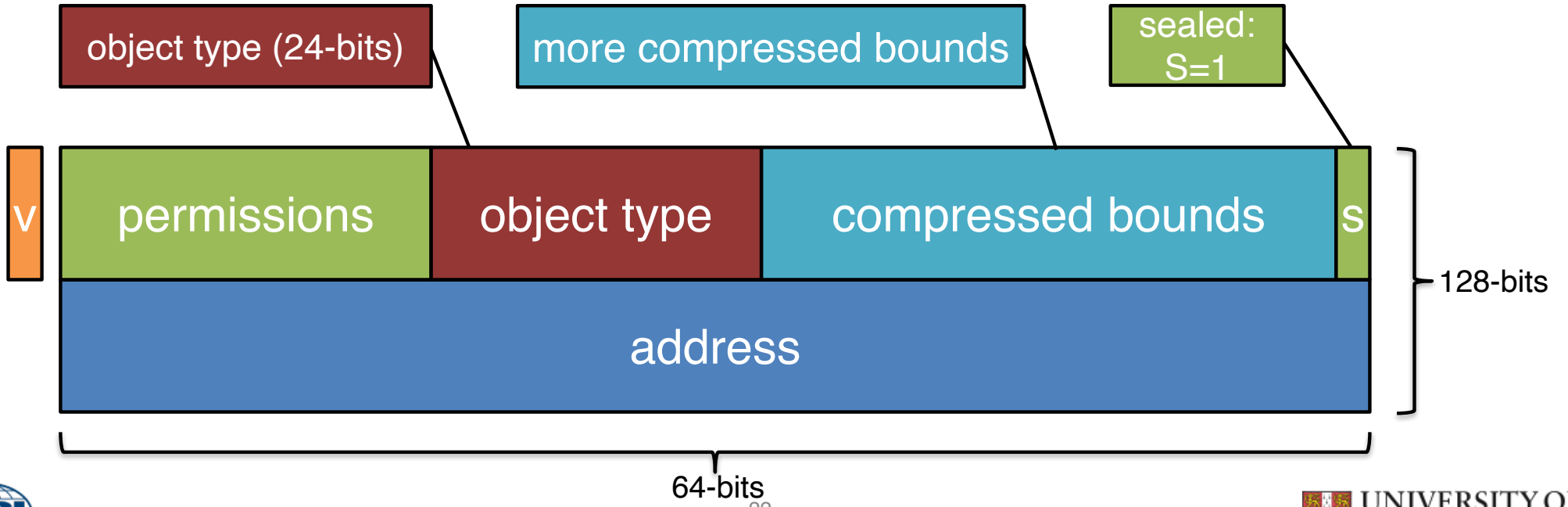
- CHERI dramatically improves **compartmentalization scalability**
  - More compartments
  - More frequent and faster domain transitions
  - Faster shared memory between compartments

Benchmarks show a 1-to-2 order of magnitude performance inter-compartment communication improvement compared to conventional designs

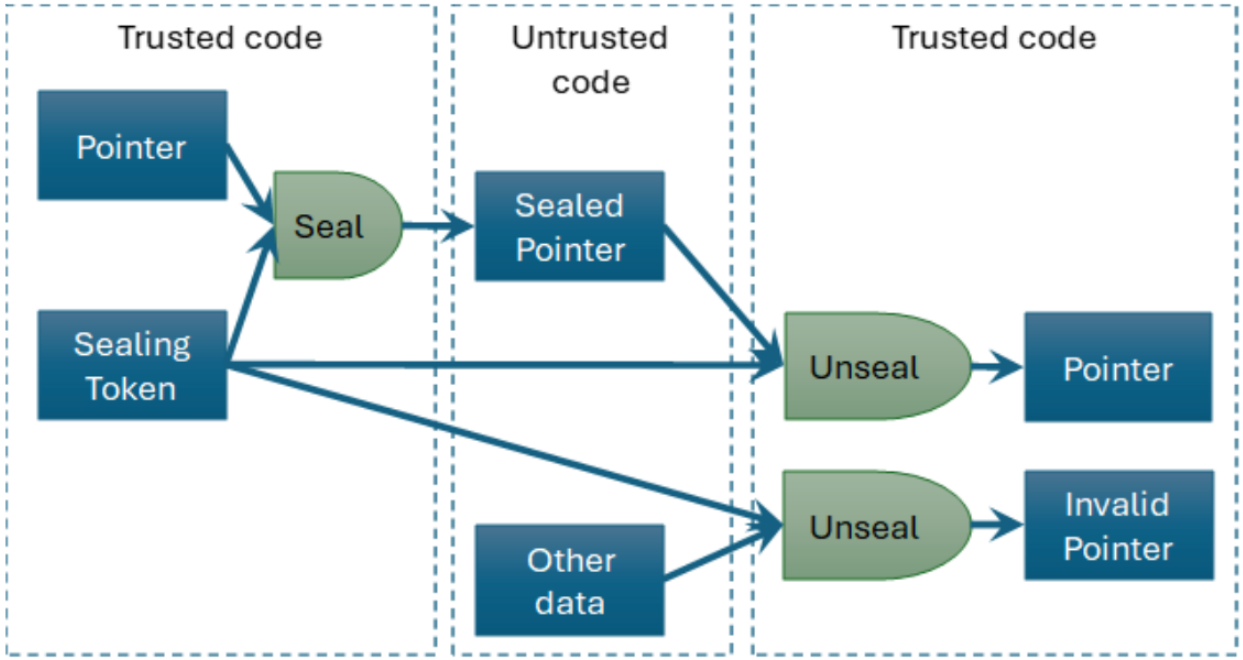
- Many potential use cases – e.g., sandbox processing of each image in a web browser, processing each message in a mail application
- Unlike memory protection, software compartmentalization requires **careful software refactoring** to support strong encapsulation, and affects the software operational model

# Sealed Capabilities for Compartmentalization

- Sealed capabilities are non-dereferencable capabilities
- Have to be unsealed (e.g. inside a compartment) before use



# Passing Capabilities via Untrusted Code




- The untrusted code cannot use a sealed capability.
- Sealing and unsealing requires a “sealing token” or capability.

# Two promising CHERI-based compartmentalization models

**Library compartmentalization (c18n)**

- Reuse existing library (or library subset) abstractions
- Existing memory-safe linkage as foundation
- Interpose domain-transition trampolines on PLT calls
- Separate per-compartment stacks
- Policy language enables sub-library boundaries to be used

Shipped in CheriBSD 22.12.

Updates in 23.11, 24.05, and 25.03.

**UNIX co-located processes (co-processes)**

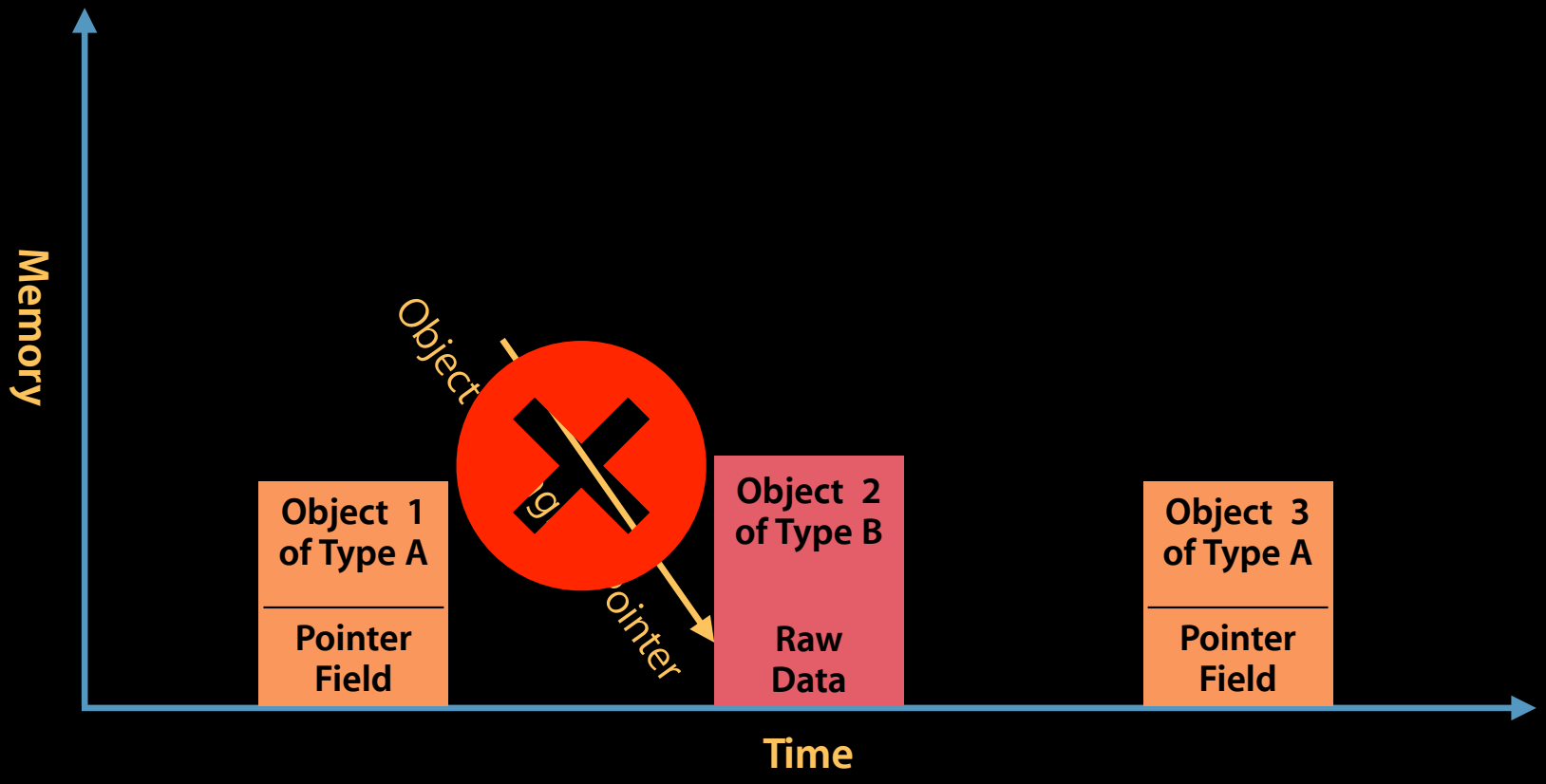
- Multiple processes in each address space, separate by CHERI
- Kernel protrudes fast domain-transition switcher into userlevel
- Inter-process shared memory shares TLB entries
- Measured 1-2 orders of magnitude performance vs. IPC

Prototype running; targeted for inclusion in a 2026 CheriBSD release.

# CHERI Temporal Memory Safety

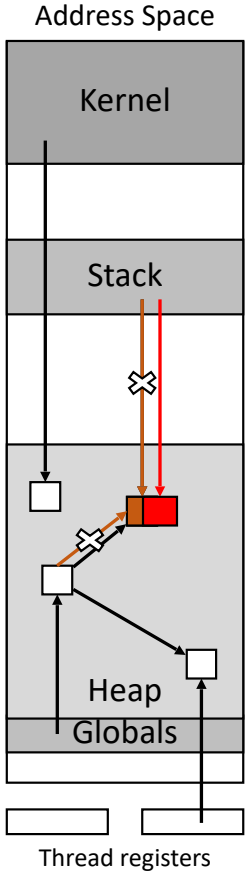
# Sweep and Revoke Dangling Pointers

## Assumption: Dangling pointers can be easily found and disabled



# Cornucopia: CHERI Heap Temporal Safety

## Address Space Quarantine, Revocation




- Focused on *heap* temporal safety
  - More complex lifetimes than stack objects, resists static approaches
- Heap pointers end up in globals, stacks, registers, kernel heap, …

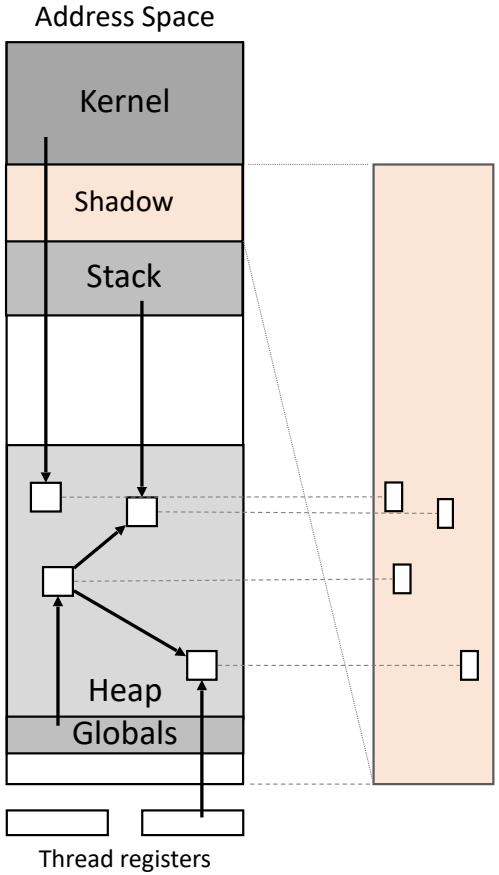
- Risk: retain references to `free()` object, overlap new allocation
  - Use After Reallocation: use old reference to access new allocation
  - UAF-but-not-UAR less of a concern

- Eliminate UAR by *revoking* dead references
  - UAF left possible, but guaranteed to access old object

- "Dual" of garbage collection: (lazily by quarantine) enforce `free()`

# Cornucopia: CHERI Heap Temporal Safety

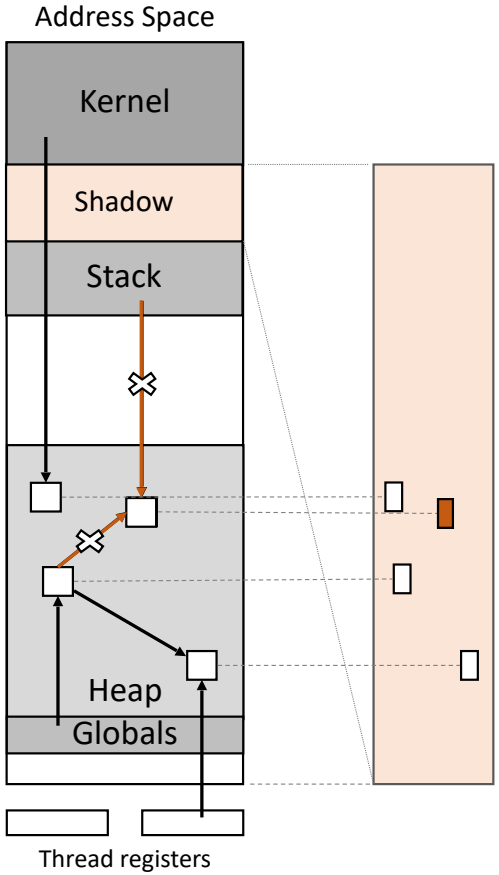
## Kernel Revocation Service




- Kernel offers revocation *service* to user programs
- Exposes “shadow bitmap”
  - Encodes live/free state of memory, 1 bit per 16 bytes
- Deletes capabilities *to* addresses with set bits
  - Promises to inspect itself as well as user program
- Thread-safe & mostly concurrent implementation

# Cornucopia: CHERI Heap Temporal Safety

## Quarantine & Batched Revocation




- On free, allocator…
  - *marks* shadow of object
  - holds address space in *quarantine*
- When quarantine fills, allocator invokes revoker service
  - Deletes all capabilities whose targets have marked shadows
- After revocation, safe to reuse address space
  - Allocator *clears* shadow, enqueues address space to free lists

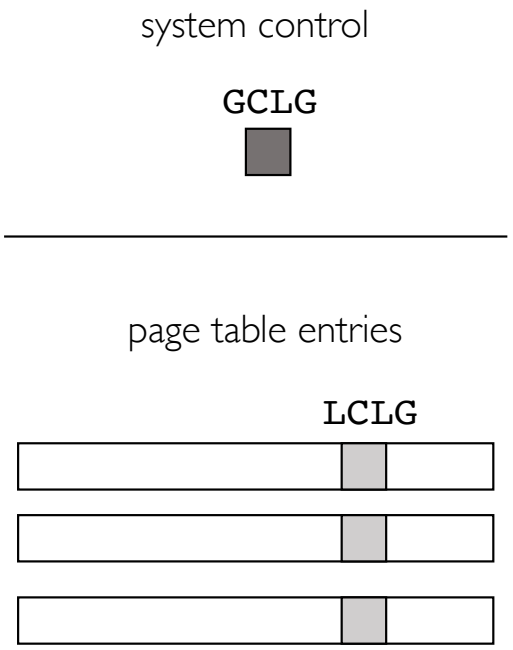
# Cornucopia: CHERI Heap Temporal Safety

## Additional hardware to improve revocation sweep performance

- Revocation sweeps are done in a batch when quarantine buffer is full. (Less disruptive on cache, etc.)
- Capability-dirty bit in the Page Table Entry (PTE) which is set when a capability is stored to a page. (Only *capability-dirty* pages are scanned.)
- CLoadTags instruction returns a bit map of capabilities in a cache line without loading memory into cache. (Skip loading clean cache lines.)
- Load Barrier. An idea inspired by garbage collectors….

# Novel Architectural Support: Capability Load Generations

- The system has a global control bit called **GCLG** (Global Capability Load Generation)
- PTEs have a **LCLG** (Local Capability Load Generation) bit
- When a PTE mapping experiences a capability load, if its **LCLG** value does not equal the system's **GCLG** value, the load faults
  - Steady state: all PTEs have **LCLG == GCLG**, capability loads allowed
  - When **GCLG** flipped, capability loads via all PTEs fault; fault handler can flip a PTE's LCLG and restart the load




CHERI

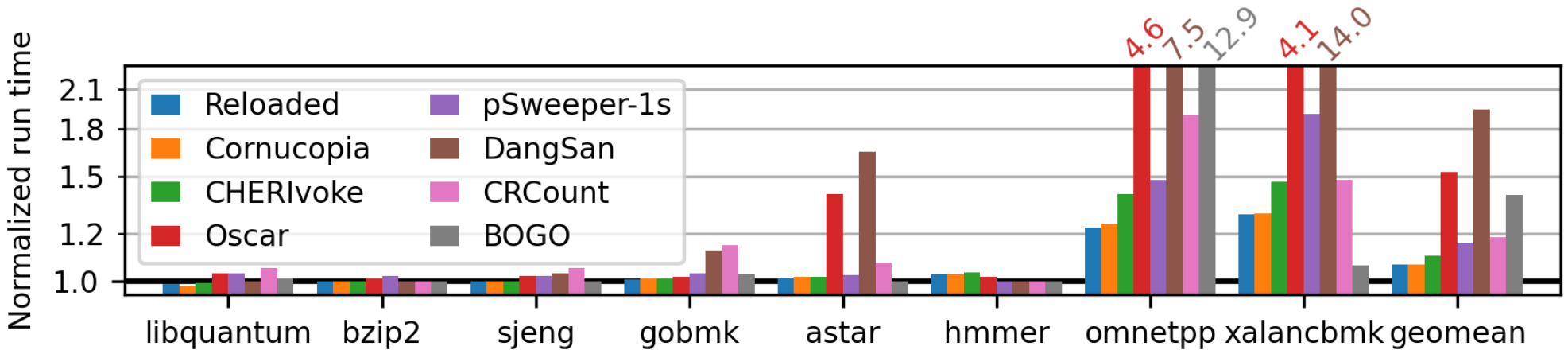
SRI UNIVERSITY OF CAMBRIDGE

# Load-Side Barrier Algorithm

- When a revocation sweep is initiated, a stop-the-world phase flips GCLG (and invalidates TLBs, sweeps the register file, userspace pointers held in the kernel, etc.)
- Revocation sweep proceeds concurrently with the application
  - Once a page has been swept, allow capability loads by flipping the PTE's **`LCLG`** bit to match **`GCLG`**
- Capability load faults experienced by the application cause synchronous processing for revocation and PTE updates as needed

# Evaluation: Wall-Clock Overhead

- Wall-clock overheads versus prior CHERI work (re-implemented and measured) and other published techniques for heap temporal safety (using previously published results), SPEC CPU2006 (lower is better)




CHERI

SRI

UNIVERSITY OF CAMBRIDGE

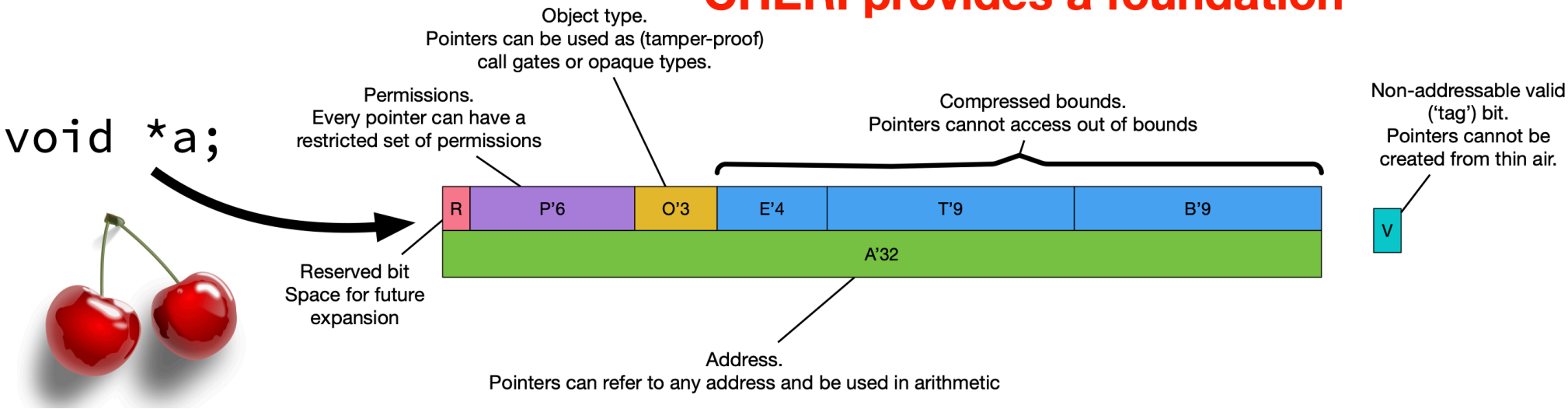
# CHERI Temporal Memory Safety Papers

- CHERIvoke (Xia et al., MICRO 2019)
  - Batch revocation sweeps using a quarantine buffer of memory that has been freed by the application but not returned to the allocator for reuse and a bitmap to indicate which regions of memory are subject to revocation
  - Introduced and used architecture-supported capability-dirty tracking via page table entry (PTE) bits that become set when a mapping experiences a capability store (skip cap-clean pages)
  - Introduced and used the **CLoadTags** instruction, which allows a cache line to be scanned for valid capabilities without loading the underlying data (skip loading cap-clean cache lines)
- Cornucopia (Filardo et al., IEEE SP 2020)
  - Fully elaborated CheriBSD kernel subsystem for sweeping capability revocation
  - Novel concurrent store-side barrier sweeping algorithm that improved upon CHERIvoke by using the capability-dirty tracking bit to support concurrency
  - Significantly reduced pause times and execution times (when sweeping in parallel) relative to CHERIvoke
- Cornucopia Reloaded (Filardo et al., ASPLOS 2024)
  - Adds a per page capability load barrier using the Morello prototype.
  - Nearly eliminates application pause times.

# CHERIoT
# The smallest variety of CHERI

# CHERIoT: 32 Bit CHERI

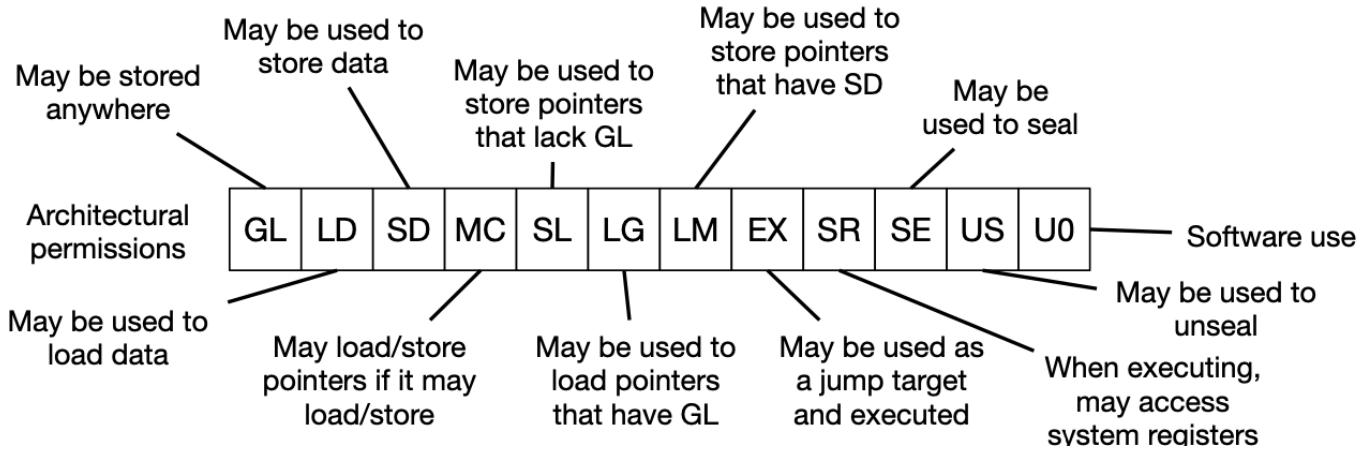
**CHERI provides a foundation**



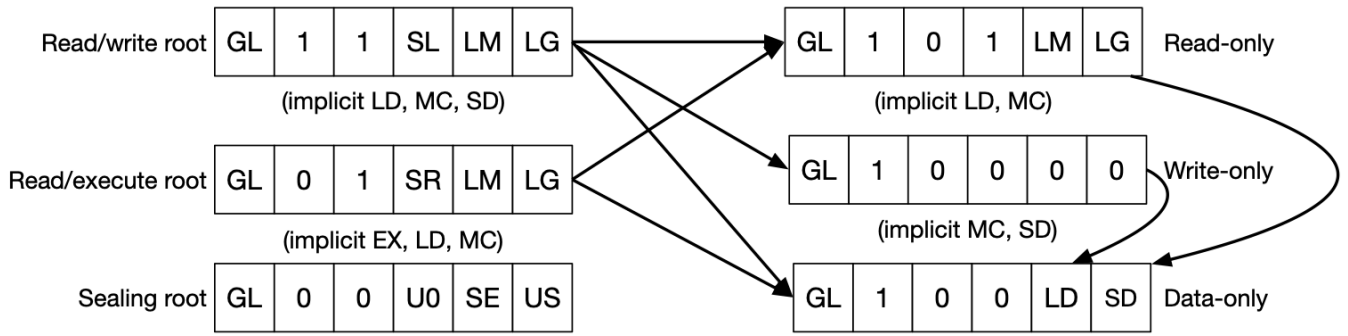

- Optimized for IoT/embedded applications.
- Pure capability mode only (when booted in CHERI mode). No hybrid mode.
- Physical memory only. No MMU.
- Only adds about 30 instructions to RV32IMCB with the Integer (I), Integer Multiplication and Division (M), Compressed (C), and Bit Manipulation (B) instruction extensions.

# Compressed Permission Encoding

**Architectural Permissions**
**12 Perm Bits**

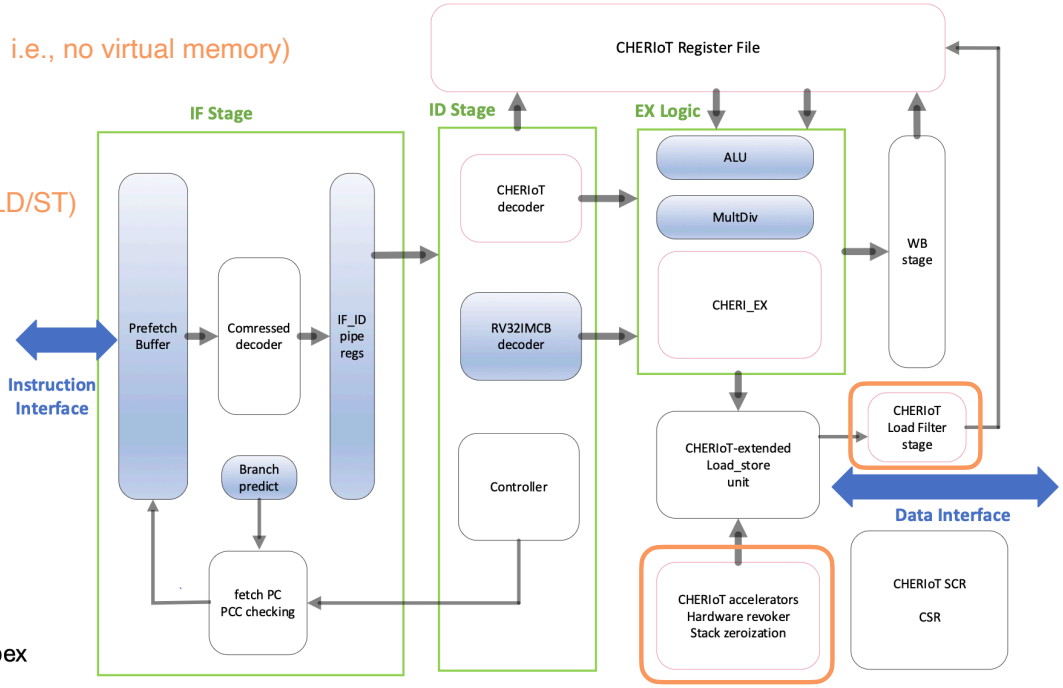



**Compressed Permissions**
**6 Bits**



# The CHERIOT-Ibex design

- Ibex core as the starting point
  - Small 32-bit, 3-stage pipeline RV32IMCB core (No MMU. i.e., no virtual memory)
  - Open-source maintained by lowRISC
- CHERIOT-Ibex work at Microsoft
  - Full CHERIoT ISA support (Requires two 33-bit LD/ST)
    - All instructions 1-cycle except for CLC/CSC
  - Capability-extended register file
  - Extended 33-bit data load/store interface
    - MSB bit carries capability tag
  - Bounds/permission checking and fault handling
    - for all data accesses and instruction fetches
  - SCR (special capability registers)
  - HW assistance for RTOS
    - Capability load filtering and background revocation
    - Stack zeroization for compartment switching
  - RV32IMCB backward compatibility mode
    - Run-time selectable, LEC verified with the original Ibex
  - CHERIoT-aware RISC-V debugging support

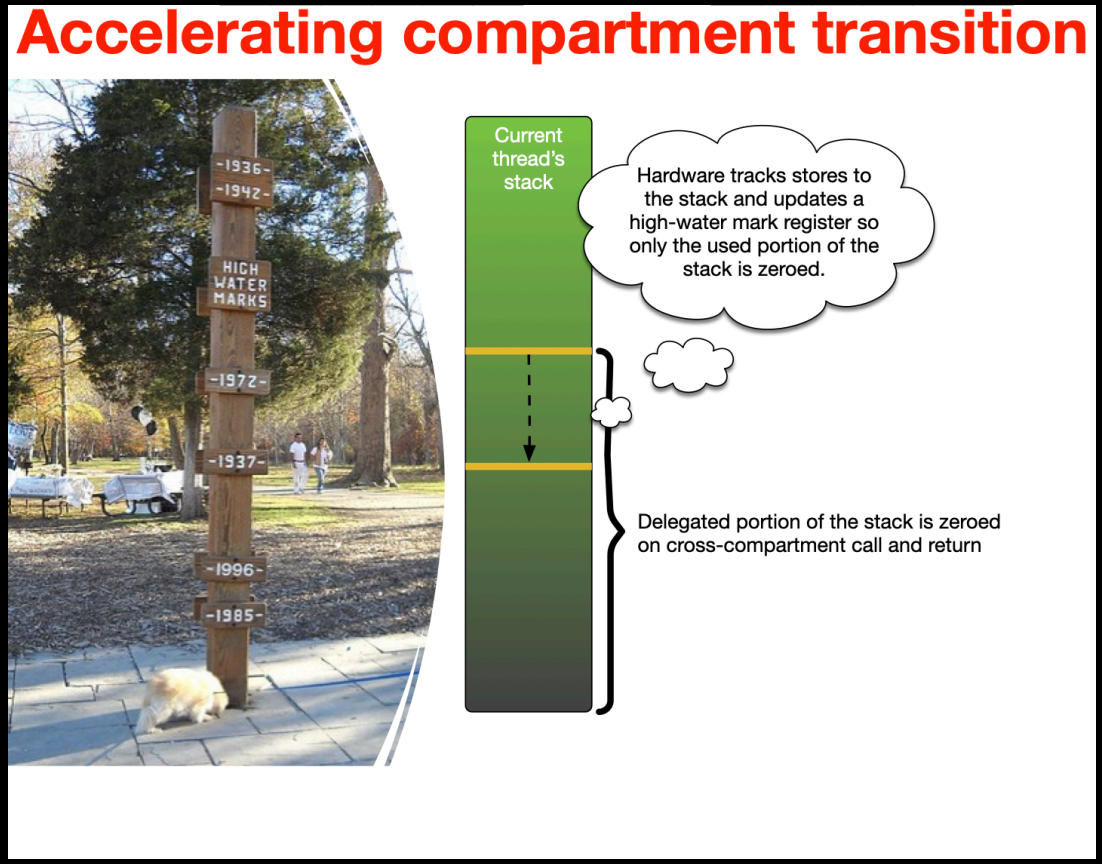


# CHERIoT HW Assistance for RTOS

- Stack Zeroization for compartment switching
  - Zero stack before calling compartment and after it returns
- Capability Load Filtering
  - Implements the load-side barrier in a simplified way
- Background Capability Revoker
  - Sweeps memory for capabilities referencing freed memory regions and revokes them

# CHERIoT Local-Global and Stack Clearing

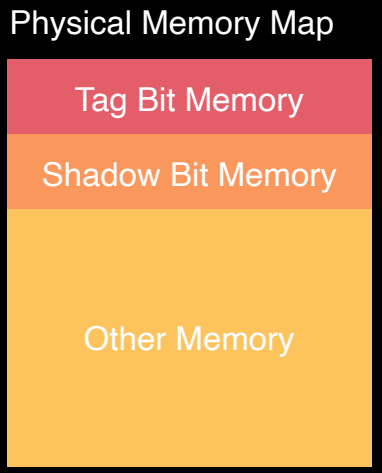
- Cross-compartment calls: caller strips the G permission from all objects that it wants to temporarily delegate. These capabilities can only be stored on the stack.
- The switcher provides the callee with a program stack (part of the caller's stack).
- When callee finishes and returns, the provided stack is zeroed (using HW stack zeroization mechanism). Callee will no longer have access to the delegated capabilities.



# CHERIoT Load-Filter Hardware

**Enforces no loading of valid capabilities pointing to freed memory**

- Capability loads do two 33-bit memory loads:
  - Capability meta data + tag bit
  - Address pointer + (heap) shadow bit
- Shadow (freed) bits are accessed through a dedicated memory interface (similar to tag bits).
- One shadow bit for every 8 bytes (same granularity as tags).
- If the shadow bit is set then tag bit is cleared (capability is invalidated).



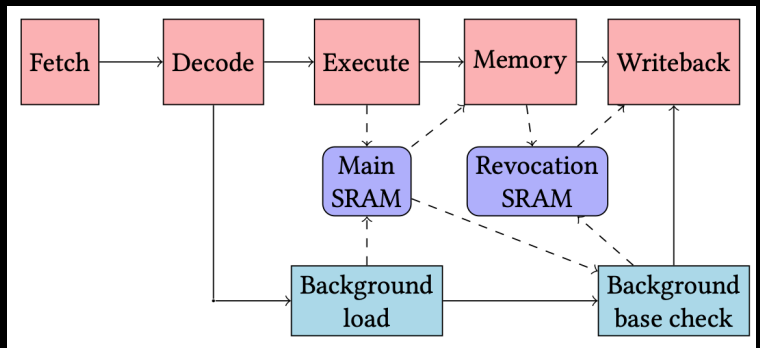




**Figure 4: Hardware load filter in a 5-stage pipeline. Arrows indicate pipeline flow whereas dotted arrows indicate SRAM requests and responses.**

# CHERIoT Capability Revocation

1. Set shadow (freed) memory tags for revoked memory.
2. Start hardware revoker sweeping memory region for capabilities to revoke.
3. While sweeping invalidate capabilities that are loaded pointing to freed memory.
4. After sweeping clear shadow (freed) memory tags.

**Bus Contention Priority:**

1. CPU Pipeline (Highest)
2. Stack zerorization engine
3. Background revocation engine

(Embedded apps typically spend less than 50% of its CPU cycles reading or writing)

**Background Revocation Registers:**

- **Start**: Start of Region
- **End**: End of Region
- **Epoch**: RO revocation status
- **Kick**: WO start revocation engine

# CHERIoT-Ibex Timing, Area, and Power

## Compared to the original Ibex (w/ 16PMP)

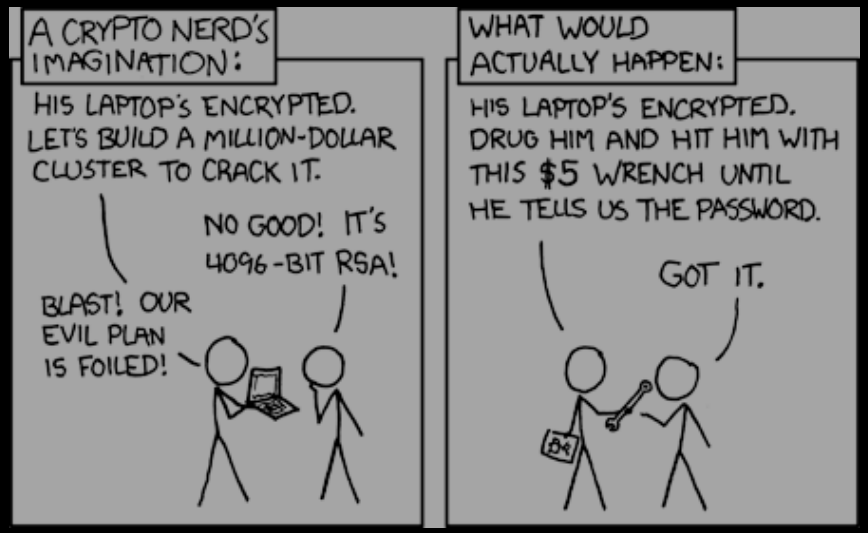
- Evaluation at 5nm (TSMC 5fpp) and 28nm (TSMC 28LP) process node.
- Fmax essentially same
- Area increase of ~10%
- Power consumption similar
  - LVTLL (Low-voltage TTL)
  - LVT (Low-voltage transistor)
  - ULVT (Ultra-low-voltage transistor)

| 5ffp results | | CHERIOT-Ibex | Ibex (16PMP) |
|---|---|---|---|
| Frequency | | 550MHz | 550MHz |
| Area | | 2028.5 | 1900.38 |
| Total Power | | 0.55mW | 0.76mW |
| VT Mix | LVTLL | 67.24% | 62.29% |
| | LVT | 18.73% | 26.75% |
| | ULVT | 14.04% | 10.96% |

# The End…

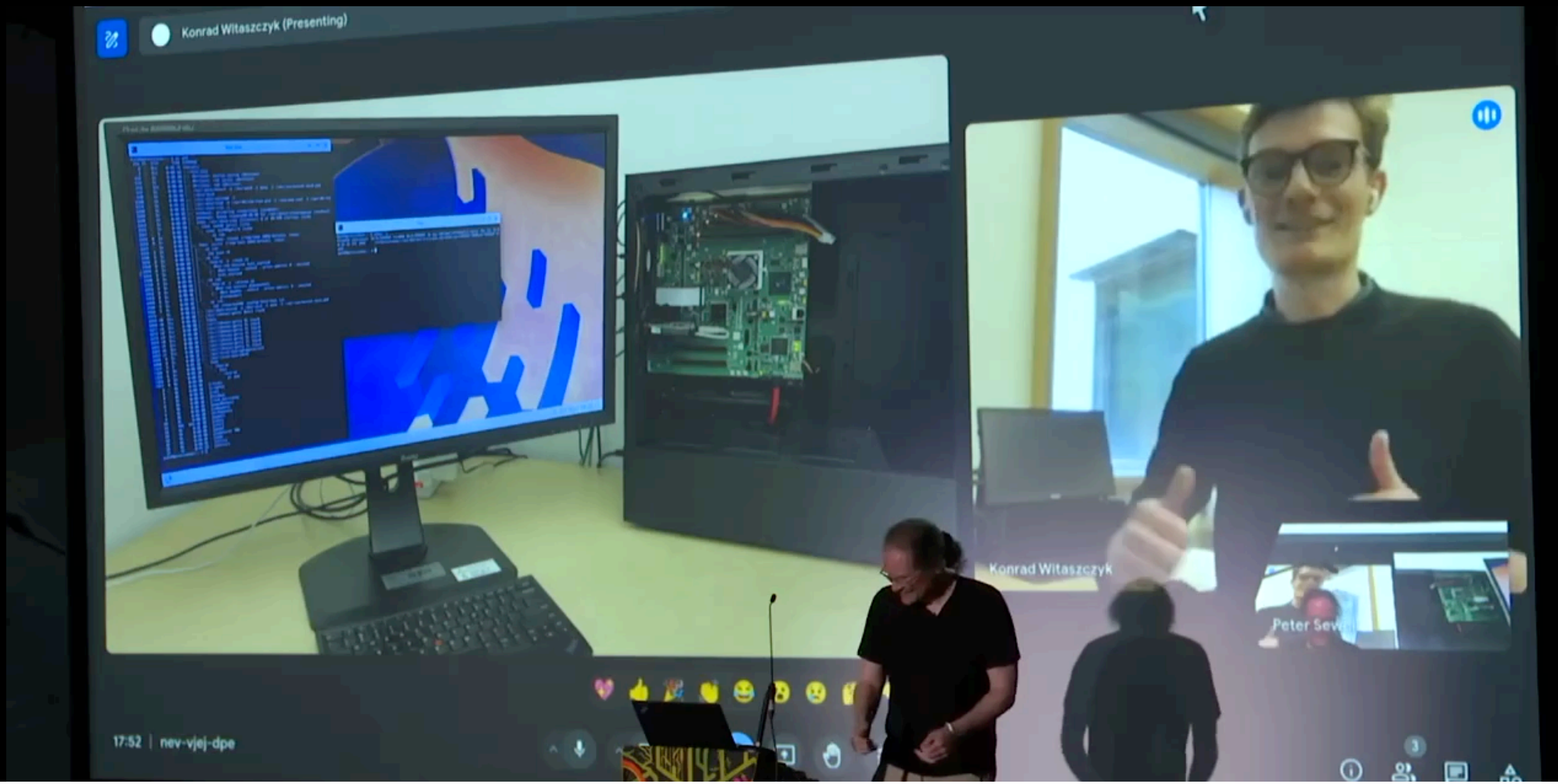
For more information:

- HW Security links:
  - https://hwsec.son.org/
- My Email: sson at me dot com



Let’s see some demos!

# CHERI Morello Demo: Memory Protection



# CHERI Morello Demo: Compartmentalization

# CHERIoT Demo: Compartment Fault and Recovery

## Memory safety bug added to TCP/IP stack